ADT-8933

三轴运动+激光雕刻打标控制卡

说

明

书



深圳市众为兴数控技术有限公司

地址:深圳市南山区马家龙工业区 36 栋 5 楼　邮编:518052

电话:0755-26722719（20 线）　传真:0755-26722718

版权声明

商标声明

版本升级说明

版本号	修改日期	说明
V100	2007/03	第一版

目　　录

第一章 概要

☞ 产品简介

ADT8933 卡是基于 PCI 总线的高性能三轴伺服/步进+激光雕刻打标控制卡，一个系统中可支持多达 16 块控制卡，可控制 48 路伺服/步进电机，支持即插即用.。

脉冲输出方式可用单脉冲（脉冲+方向）或双脉冲（脉冲+脉冲）方式，最大脉冲频率 2MHz，采用先进技术，保证在输出频率很高的时候，频率误差小于 0.1%。

支持任意 2-3 轴直线插补，最大插补速度 1MHz。

速度控制可用定速和梯形加减速。

位置管理采用两个加/减计数器，一个用于内部管理驱动脉冲输出的逻辑位置计数器，一个用于接收外部的输入，输入信号是 A/B 相输入的编码器或光栅尺，作为实际位置计数器

计数器位数高达 32 位，最大范围–2,147,483,648~+2,147,483,647。

提供一路 PWM 输出和三路 8 位 D/A 输出，可用于激光强度控制。

提供 DOS、WINDOWS95/98/NT/2000/XP 开发库，可用 VC++、VB、 BC++、LabView 等进行软件开发。

☞ 主要性能

- 32 位 PCI 总线，即插即用
- 所有输入、输出均采用光耦隔离,抗干扰性强
- 3 轴伺服/步进电机控制，既可独立控制，又可联动和插补控制
- 脉冲输出的频率误差小于 0.1%
- 最大脉冲输出频率为 2MHz
- 脉冲输出可用单脉冲（脉冲+方向）或双脉冲（脉冲+脉冲）方式
- 一路编码器反馈输入，32 位计数，最大计数范围 -2,147,483,648~+2,147,483,647
- 梯形加/减速
- 2-3 轴直线插补
- 最大插补速度 1MHz
- 运动中可以实时读出逻辑位置、实际位置、驱动速度
- 10 路数字输入，5 路数字输出
- 支持在一个系统中使用多达 16 个控制卡
- 支持 DOS、WINDOWS95/98/NT/2000/XP 等操作系统

☞ 应用范围

- 激光雕刻系统
- 激光打标系统
- 基于 PC 的三轴数控系统

第二章 硬件安装

☞ 配件

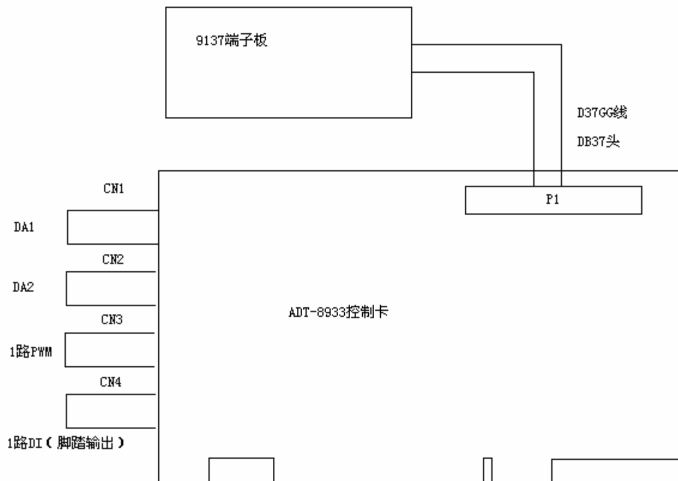
1. ADT-8933 **用户手册（本手册）**
2. ADT-8933 **三轴** PCI **总线运动控制卡**
3. ADT-8933 **用户光盘**
4. ADT-9137 37 **芯信号接线板** 1 **块**
5. ADT-D37　37 **芯屏蔽连接线** 1 **条**
6. ADT-DB37 37 **芯扁平线** 1 **条**

☞ 安装

1. **关闭电脑电源（注：**ATX **电源需总电源关闭）。**
2. **打开电脑机箱后盖。**
3. **选择一条未占用的** PCI **插槽,插入** ADT-8933**。**
4. **确保** ADT-8933 **的金手指完整插入** PCI **插槽,拧整螺丝。**

第三章 电气连接

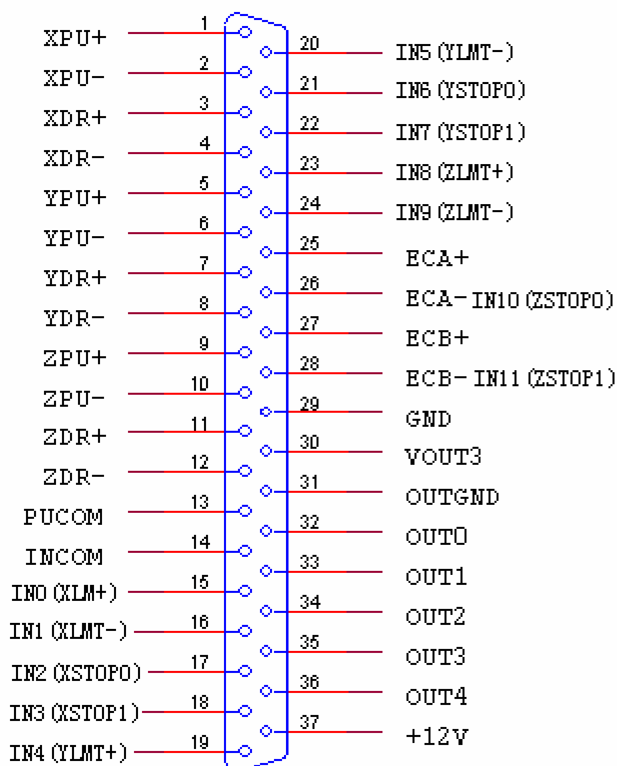
ADT_8933 **卡的电气连接图如下：**



一块 Adt8933 **卡有** 5 **个输入/输出接口，其中** P1 **为** 37 **针插座，**CN1、CN2 **为两路** DA **输出，**CN3 **为一路** PWM **输出，**CN4 **为一路** DI**，用于控制脚踏输出。针插座。**

P1 为 X、Y、Z **轴的脉冲输出、开关量输入** IN0-IN9、**开关量输出** OUT0-OUT4 **和一路** DA **输出的信号接线。**

☞ P1 线号说明



线 号	符 号	说 明
1	XPU+/CW+	X 脉冲信号+
2	XPU-/CW-	X 脉冲信号-
3	XDR+/CCW+	X 方向信号+
4	XDR-/CCW-	X 方向信号-
5	YPU+/CW+	Y 脉冲信号+
6	YPU-/CW-	Y 脉冲信号-
7	YDR+/CCW+	Y 方向信号+

8	YDR-/CCW-	Y 方向信号-
9	ZPU+/CW+	Z 脉冲信号+
10	ZPU-/CW-	Z 脉冲信号-
11	ZDR+/CCW+	Z 方向信号+
12	ZDR-/CCW-	Z 方向信号-
13	PUCOM	用于单端输入的驱动器　不可接外接电源
14	INCOM	光耦输入公共端（下面的信号）
15	IN0	X 正限位，可用作通用输入信号 0
16	IN1	X 负限位，通用输入信号 1
17	IN2	通用输入信号 2
18	IN3	通用输入信号 3
19	IN4	Y 正限位，可用作通用输入信号 4
20	IN5	Y 负限位，可用作通用输入信号 5
21	IN6	通用输入信号 6
22	IN7	通用输入信号 7
23	IN8	Z 正限位，可用作通用输入信号 8
24	IN9	Z 负限位，可用作通用输入信号 9
25	XECA+	X 轴编码器 A 相输入+
26	XECA-	X 轴编码器 A 相输入-， 可做通用输入信号 10
27	XECB+	X 轴编码器 B 相输入+
28	XECB-	X 轴编码器 B 相输入-，可做通用输入信号 11
29	GND	内部电源地线
30	VOUT3	第 3 路 D/A 输出
31	OUTGND	OUT0-4 开关量输出点公共负端
32	OUT0	开关量输出点
33	OUT1	
34	OUT2	
35	OUT3	
36	OUT4	

37	+12V	内部+12V 电源正端　不可接外接电源

说明：编码器用作通用输入信号时，XECA+、XECB+分别用作对应输入信号的公共端。公共端电压必须使用+5V，如果用外部+12V 电源，必须串 1K 电阻。

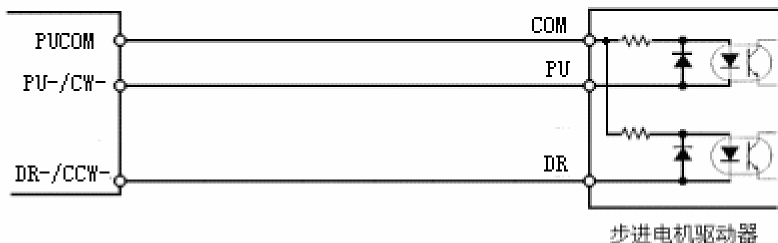
具体接线方法参照下面数字输入连接部分。

☞　脉冲/方向输出信号的连接

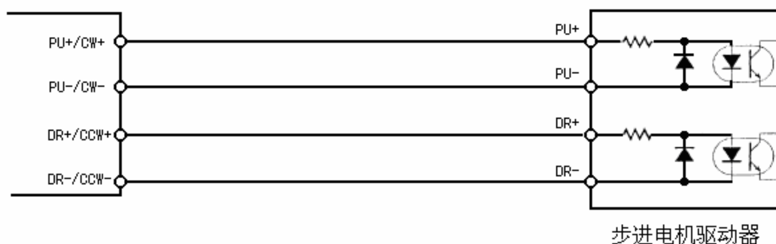
脉冲输出为差动输出方式

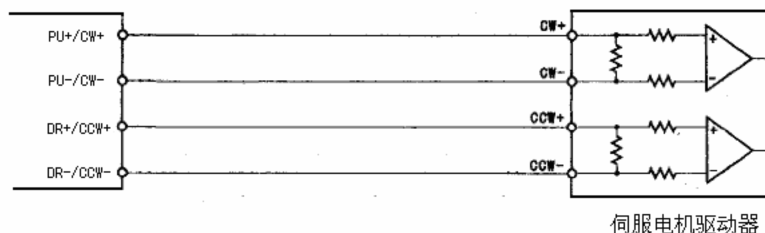
可与步进/伺服驱动器很方便的连接

下图为脉冲与方向的阳极已连通的接法



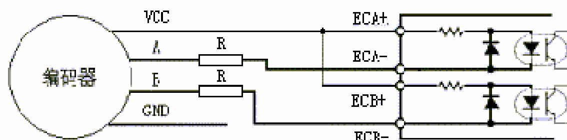
下图为脉冲与方向信号独立的接法，建议采用此种方法，因为是差动接法，抗干扰性强。





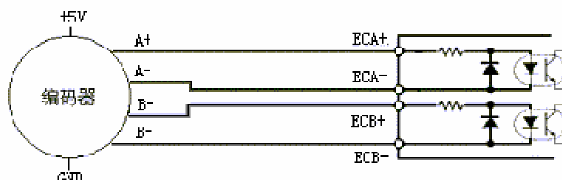
说明：步进电机驱动器、常用伺服电机驱动器和端子板的接线图参见附录 A。

☞ 编码器输入信号的连接



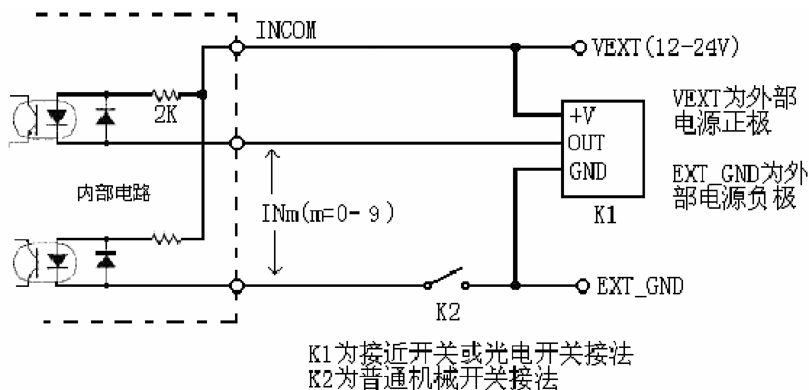
集电极开路（OPEN-COLLECT）输出型编码器接线图

+5V电源时 R可不用 +12V电源时 R=1KΩ +24V电源时 R=2KΩ



差分驱动（LINE DEIVER）输出型编码器接线图

☞ 数字输入的连接

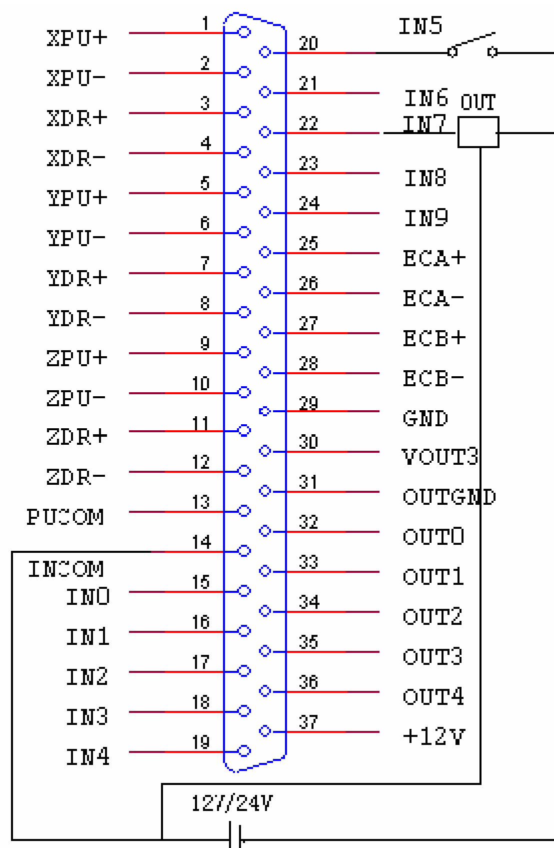


说明：

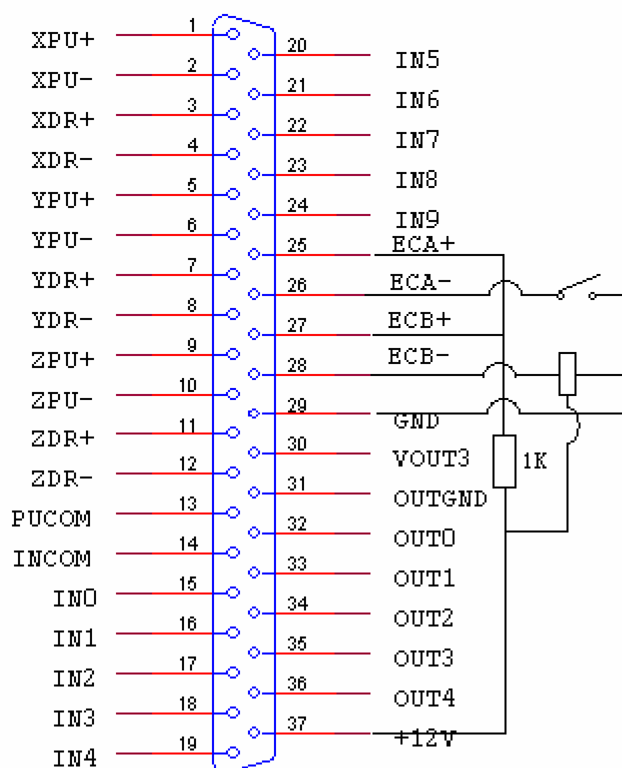
(1)IN0-IN9 的公共端为 INCOM；

(2)为了使输入信号有效，首先必须确保对应输入信号的”光耦公共端”(INCOM)已经和 12V 或 24V 的电源正端相连；其次普通开关的一端或接近开关的地线和电源负端(地线)相连；最后普通开关的另一端或接近开关的控制端必须和端子板对应的输入端相连。

(3)下面是普通开关和接近开关在使用外部电源给“光耦公共端”供电的实际接线图。

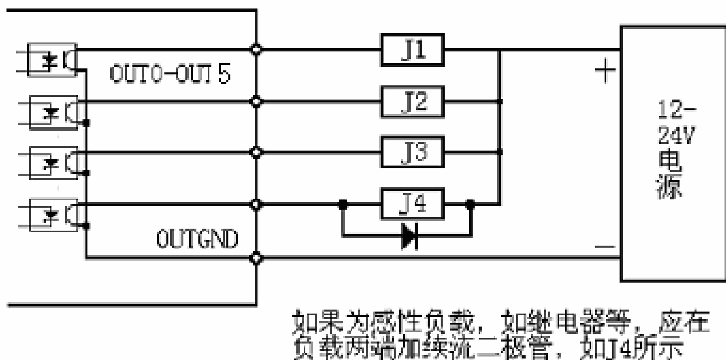


(4)**利用内部** 5V **电源给编码器“光耦正端”供电的接线图，如若电压高于** 5V**，一定要串联相应大小的电阻，**12V **串** 1K **电阻，如下图所示。使用外部电源的接线方法和普通输入点一样，参照上图即可。**



XPU+
XPU-
XDR+
XDR-
YPU+
YPU-
YDR+
YDR-
ZPU+
ZPU-
ZDR+
ZDR-
PUCOM
INCOM
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
ECA+
ECA-
ECB+
ECB-
GND
VOUT3
OUTGND
OUT0
OUT1
OUT2
OUT3
OUT4
+12V
1K
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

☞ 数字输出的连接

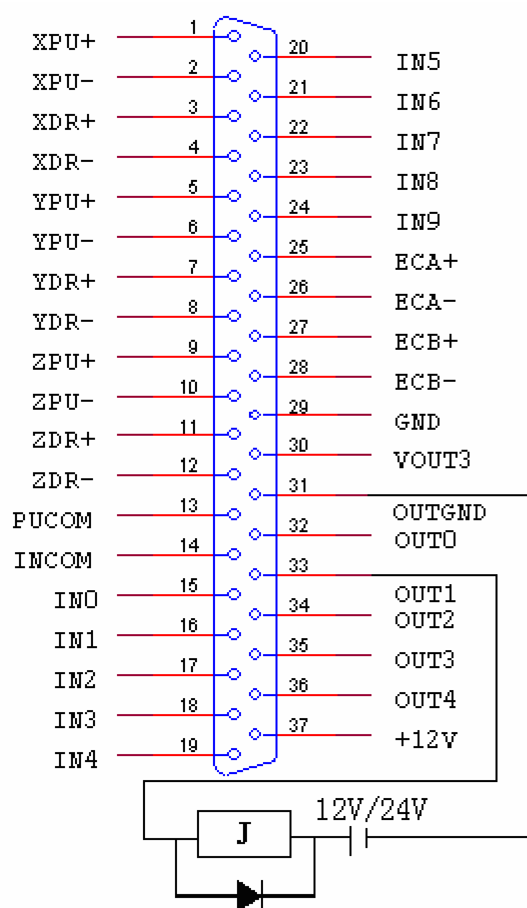


说明：

(1)OUT0-OUT5 **的公共端是** OUTGND

(2)**为了使输出信号有效，在使用外部电源时，必须确保输出公共端** OUTGND **和外部电源负端**(**地线**)**相连；在使用内部电源时，必须确保内部电源地线**(GND)**和地相连。继电器线圈的一端接电源正端；另一端接端子板对应的输出端。**

(3)**下图是继电器采用外部电源供电的实际接线图。**


XPU+
XPU-
XDR+
XDR-
YPU+
YPU-
YDR+
YDR-
ZPU+
ZPU-
ZDR+
ZDR-
PUCOM
INCOM
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
ECA+
ECA-
ECB+
ECB-
GND
VOUT3
OUTGND
OUT0
OUT1
OUT2
OUT3
OUT4
+12V
12V/24V
J

第四章 软件安装

ADT8933 卡在 Win95/Win98/NT/Win2000/WinXP 下必须安装驱动程序才能使用，在 DOS 下则无须安装驱动程序。

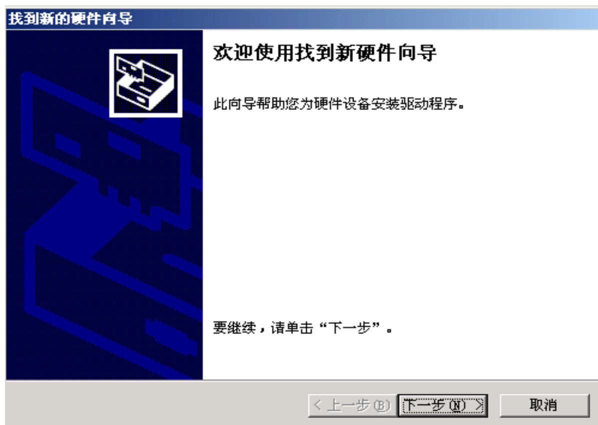
以下以 Win2000、 WinXP 为例，其余系统可参考。

控制卡驱动程序位于光盘上“开发包\驱动\控制卡驱动程序”文件夹下面，驱动程序文件名为 ADT8933.INF。

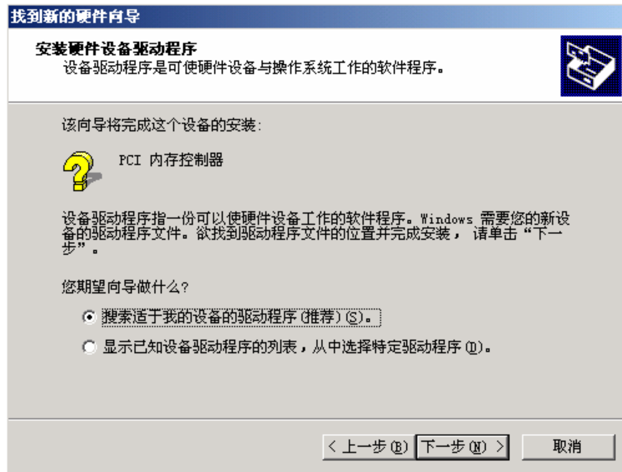
☞ Win2000 下驱动程序的安装

以下用 Win2000 Professional 中文版为例，说明驱动程序的安装，其余版本的 Win2000 与此类似。

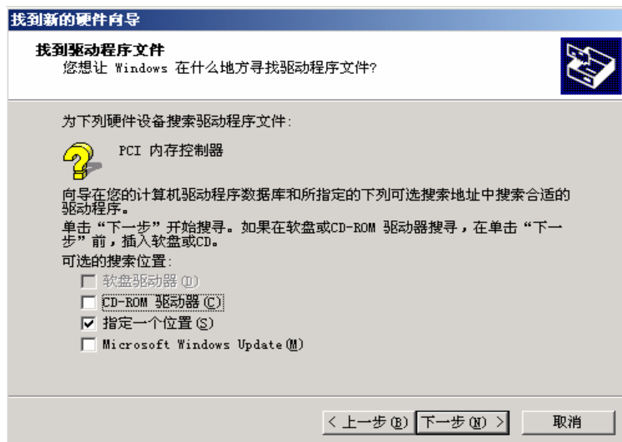
在将 ADT8933 卡安装到电脑上的 PCI 插槽后，开机时应以管理员身份登录，电脑开机后应发现新硬件，出现如下画面：



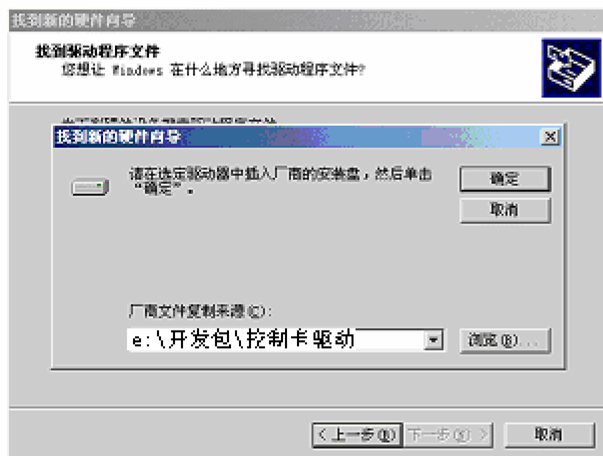
单击“下一步”后，再显示如下画面



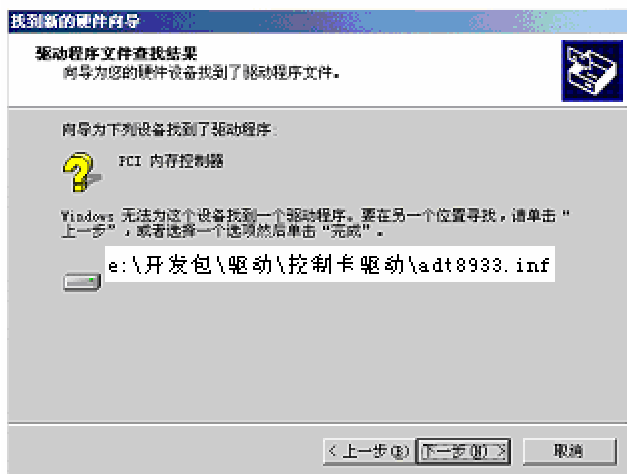
按上图选择后，再单击“ 下一步 ”后，出现如下画面



再按上图选择“指定一个位置”，单击“下一步”，出现如下画面



点击“浏览”按钮，选择光盘“开发包\驱动\控制卡驱动程序”，即可找到 ADT8933.INF **文件的路径，点击“确定”，出现如下界面**



单击"下一步"后出现如下画面

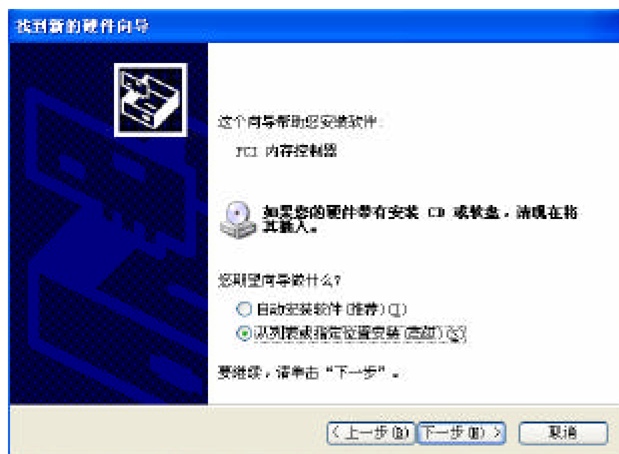


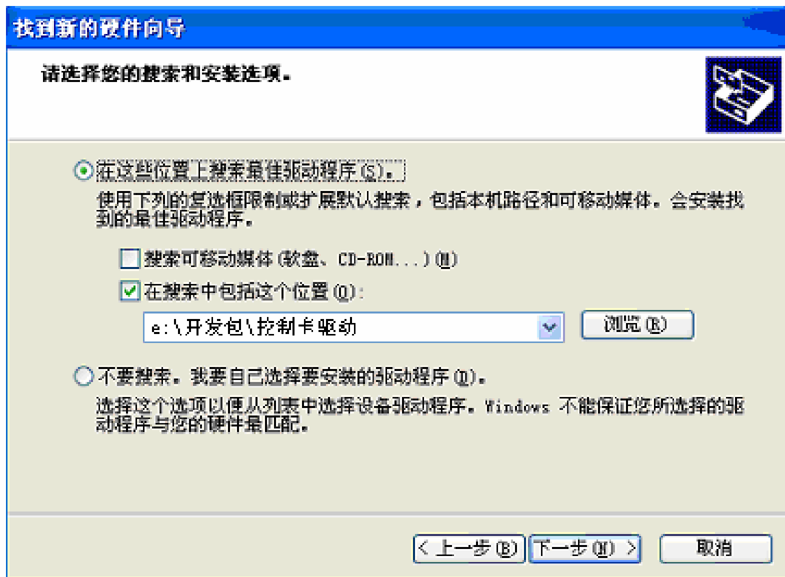
单击"完成"后，即完成 ADT8933 **卡的安装**

☞ WinXP 下驱动程序的安装

WinXP 下的安装与上面类似，参考下图：


找到新的硬件向导
欢迎使用找到新硬件向导
Windows 将通过在计算机、硬件安装 CD 或 Windows Update 网站(在您允许的情况下)上查找来搜索当前和更新的软件。
Windows 可以连接到 Windows Update 以搜索软件吗?
是，仅这一次(Y)
是，这一次和每次连接设备时(E)
否，暂时不(T)
单击“下一步”继续。
< 上一步(B)
下一步(N) >
取消


找到新的硬件向导
这个向导帮助您安装软件:
PCI 内存控制器
如果您的硬件带有安装 CD 或软盘，请现在将其插入。
您期望向导做什么?
自动安装软件(推荐)(I)
从列表或指定位置安装(高级)(S)
要继续，请单击“下一步”。
< 上一步(B)
下一步(N) >
取消


找到新的硬件向导
请选择您的搜索和安装选项。
在这些位置上搜索最佳驱动程序(S)。
使用下列的复选框限制或扩展默认搜索，包括本机路径和可移动媒体。会安装找到的最佳驱动程序。
搜索可移动媒体(软盘、CD-ROM...)(M)
在搜索中包括这个位置(O):
e:\开发包\控制卡驱动
浏览(R)
不要搜索。我要自己选择要安装的驱动程序(D)。
选择这个选项以便从列表中选择设备驱动程序。Windows 不能保证您所选择的驱动程序与您的硬件最匹配。
< 上一步(B)
下一步(N) >
取消

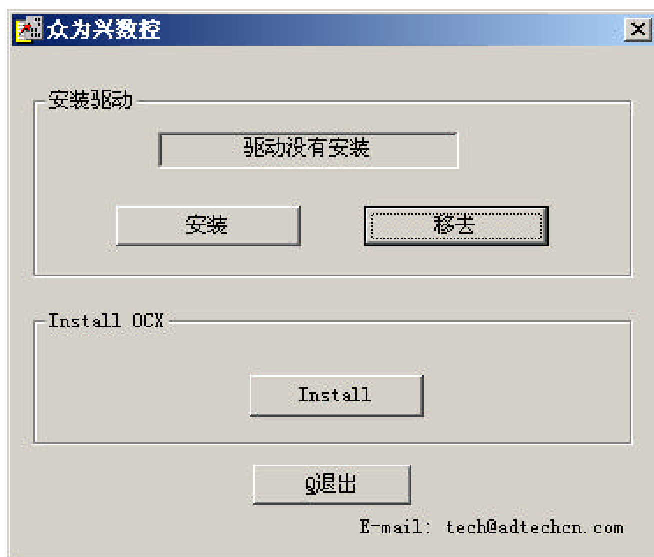




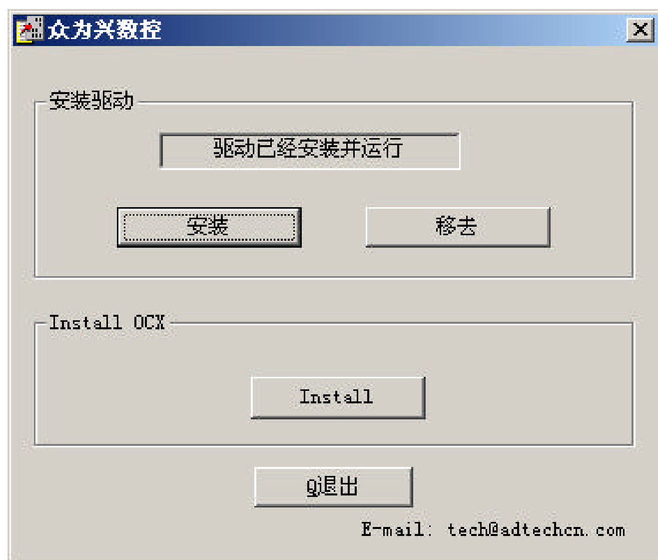
单击"完成"后，即完成 ADT8933 卡的安装

☞ Win2000&WinXP 下 IO 驱动程序的安装

操作系统平台为 Win2000 **或** WinXP **时，必须安装端口驱动。打开光盘“开发包\驱动\端口驱动程序”，驱动**.exe**”，显示如下：**



点击“安装”按钮，安装 ADT8933 **卡的** IO **驱动程序。驱动安装后，此界面变为：**



对于 Windows98 不需要运行此程序安装驱动。

☞ 动态库的安装

将光盘中"开发包\驱动\动态链接库"文件夹中的 adt8933.dll 拷贝到系统目录下,Win98 的系统目录为"Windows\System",Win2000、WinXP 的系统目录为"Windows\System32"。

第五章 功能说明

☞ 脉冲输出方式

脉冲输出有独立2脉冲和1脉冲两种方式，采用独立2脉冲方式时，正方向驱动由PU/CW 输出驱动脉冲，负方向驱动由DR/CCW输出驱动脉冲；采用1脉冲方式时，由PU/CW输出驱动脉冲，由DR/CCW输出方向信号。

脉冲/方向都是正逻辑设定时

脉冲输出方式	驱动方向	输出信号波形	
		PU/CW 信号	DR/CCW 信号
独立2脉冲方式	+方向驱动输出	脉冲	Low 电平
	-方向驱动输出	Low 电平	脉冲
1脉冲方式	+方向驱动输出	脉冲	Low 电平
	-方向驱动输出	脉冲	Hi 电平

☞ 直线插补

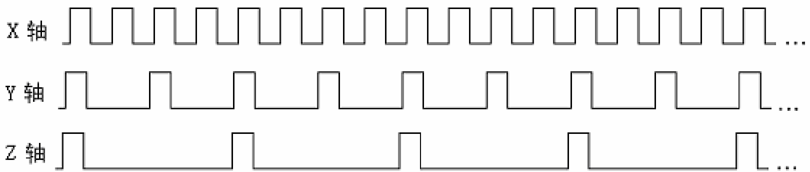
本卡可作2-3轴直线插补，支持任意2轴和任意三轴直线插补，采用改进的逐点比较法实现，可保证长轴的脉冲是均匀的，精度在一个脉冲以内。

首先取参与插补的轴中发出脉冲最多的轴，此轴即为长轴，其余轴按比例分配，速度控制只要控制长轴的速度即可。

举例说明如下(1-X轴，2-Y轴，3-Z轴)

做 3 轴直线插补

1 号轴发 2000 个脉冲，2 号轴发 1000 个，3 号轴发 500 个。



从上图可知，x 轴为长轴，其余轴按脉冲比例分配输出。

关于插补速度的设定，是以参与插补的轴中轴号最小的速度为基准，如 2、3 轴作直线插补，插补速度是由第 2 轴的速度来确定，另外请注意，插补时的速度只有单轴速度的一半，举例如下：

2、3 轴作两轴直线插补，2 轴正向发 10000 个脉冲，3 轴反向发 5000 个脉冲，即 2 号轴为长轴

```
set_startv(0,2,1000);
set_speed(0,2,1000);
inp_move2(0,2,3,10000,-5000);
```

执行如上程序后，2 号轴以 1000/2=500Hz 的频率发出 10000 个脉冲，3 号轴的频率应为 500*5000/10000=250Hz。

如果第 2 轴的速度是采用梯形加减速，则插补也是按梯形加减速运动。

第六章 ADT8933 基本库函数列表

V100 版本库函数列表

函数类别	函数名称	功能描述	页码
基本参数	adt8933_initial	初始化卡	32
	get_lib_version	获取版本号	32
	set_pulse_mode	脉冲模式	32
	set_limit_mode	限位模式	33
	set_stop0_mode	停止模式	33
	set_stop1_mode	停止模式	34
驱动状态检查	get_status	获取单轴驱动状态	34
	get_inp_status	获取插补驱动状态	34
运动参数设定	set_acc	设定加速度	36
	set_startv	设定初始速度	36
	set_speed	设定驱动速度	37
	set_command_pos	设定逻辑计数器	37
	set_actual_pos	设定实位计数器	37
运动参数检查	get_command_pos	获取逻辑位置	38
	get_actual_pos	获取实际位置	38

	get_speed	**获取驱动速度**	38
驱动类	pmove	**单轴定量驱动**	39
	dec_stop	**减速停止**	39
	sudden_stop	**立即停止**	39
	inp_move2	**两轴插补**	40
	inp_move3	**三轴插补**	40
开关量类	read_bit	**读单个输入点**	41
	write_bit	**输出单点**	41
	set_daout	DA**输出**	
	set_pwm	PWM**输出**	
延时类	set_delay_time	**设定延时时间**	
	get_delay_status	**获取延时状态**	

第七章 ADT8933 基本库函数详解

☞ 基本参数设置类

1.1 初始化卡

int adt8933_initial(void);

(1)**返回值**>0 **时，表示** adt8933 **卡的数量。如果为** 3，**则下面的可用卡号分别为** 0、1、2；

(2)**返回值**=0 **时，说明没有安装** adt8933 **卡**；

(3)**返回值**<0 **时**，-1 **表示没有安装端口驱动程序**，-2 **表示** PCI **桥存在故障。**

注意：初始化函数是调用其它函数的前提，所以必须最先调用，以确认可使用的卡数以及初始化一些参数。

1.2 获取当前库版本

int get_lib_version(); (版本为 100)

如果为 100 **则表示库版本为** 1.0.0，**依次类推。**

1.3 设置输出脉冲的工作方式

int set_pulse_mode(int cardno, int axis, int value,int logic,int dir_logic);

cardno　　**卡号**

axis　　**轴号**（1-3）

value　　0：**脉冲+脉冲方式**　　1：**脉冲+方向方式**

脉冲/方向都是正逻辑设定时

脉冲输出方式	驱动方向	输出信号波形	
		PU/CW 信号	DR/CCW 信号
独立2脉冲方式	+方向驱动输出		Low 电平
	-方向驱动输出	Low 电平	
1脉冲方式	+方向驱动输出		Low 电平
	-方向驱动输出		Hi 电平

logic 0： **正逻辑脉冲** 1： **负逻辑脉冲**

正逻辑脉冲 ： 负逻辑脉冲 ：

dir-logic 0：**方向输出信号正逻辑** 1：**方向输出信号负逻辑**

dir_logic	正方向脉冲输出时	负方向脉冲输出时
0	Low	Hi
1	Hi	Low

返回值 0：**正确** 1：**错误**

默认模式为：脉冲+方向，正逻辑脉冲，方向输出信号正逻辑

1.4 设定正/负方向限位输入nLMT信号的模式设定

int set_limit_mode(int cardno, int axis, int v1,int v2,int logic);

cardno **卡号**

axis **轴号**（1-3）

v1 0：**正限位有效** 1：**正限位无效**

v2 0：**负限位有效** 1：**负限位无效**

logic 0：**低电平有效** 1：**高电平有效**

返回值 0：**正确** 1：**错误**

默认模式为：正、负限位低电平有效

1.5 设定stop0输入信号的模式设定

int set_stop0_mode(int cardno, int axis, int v,int logic);

cardno **卡号**

axis　　轴号（1-3）

v　　0：无效　　1：有效

logic　　0：低电平有效　　1：高电平有效

返回值　　0：正确　　1：错误

默认模式为：无效

1.6 设定stop1输入信号的模式设定

int　set_stop1_mode(int cardno,　int axis,　int v,int logic);

cardno　　卡号

axis　　轴号（1-3）

v　　0：无效　　1：有效

logic　　0：低电平有效　　1：高电平有效

返回值　　0：正确　　1：错误

默认模式为：无效

☞ 驱动状态检查类

2.1 获取各轴的驱动状态

int　get_status(int cardno,int axis,int *value)

cardno　　卡号

axis　　轴号（1-4）

value　　驱动状态的指针

　　0：驱动结束

　　非 0：正在驱动

返回值　　0：正确　　1：错误

2.2 获取插补的驱动状态

int　get_inp_status(int card,int *value)

cardno　　卡号

value 插补状态的指针

　　0：插补结束　　1：正在插补

返回值 0：**正确** 1：**错误**

☞ 运动参数设定类

注意：以下参数在初始化后值不确定，使用前必须设定

3.1 加速度设定

int set_acc(int cardno,int axis,long value);

cardno **卡号**

axis **轴号**

value **范围** (1-65536))

返回值 0：**正确** 1：**错误**

加速度是直线加减速驱动中速度变化参数，加速度设定值为A，**加速度是下述算式：**

加速度（PPS/SEC）=A*125

即 加速度（PPS/SEC）=A*125

加速度设定值A **的设定范围是**1~65536。

如：

set_acc(0,1,100);

则加速度为：

100*125=12500 PPS/SEC

3.3 初始速度设定

int set_startv(int cardno,int axis,long value);

cardno **卡号**

axis **轴号**

value **范围**(**最大**2M)

返回值 0：**正确** 1：**错误**

它是加/ **减速驱动的加速开始时的速度和减速结束时的速度，初始速度设定数值为**SV **的话，初始速度是下述算式**

初始速度（PPS）=SV

3.4 驱动速度设定

int set_speed(int cardno,int axis,long value);

cardno　卡号

axis　轴号

value　范围(最大2M)

返回值　0：正确　　　　1：错误

它是加/减速驱动中达到定速区域的速度。定速驱动从该速度开始运行。

驱动速度（PPS）=V

3.5 逻辑位置计数器设定

设定逻辑位置计数器的数值

int set_command_pos(int cardno,int axis,long value);

cardno　卡号

axis　轴号

value　范围(-2147483648~+2147483647)

返回值　0：正确　　　　1：错误

逻辑位置计数器任何时候都能进行读写操作

3.6 实际位置计数器设定

设定实际位置计数器的数值

int set_actual_pos(int cardno,int axis,long value);

cardno　卡号

axis　轴号(axis=1)

value　范围(-2147483648~+2147483647)

返回值　0：正确　　　　1：错误

实际位置计数器任何时候都能进行读写操作



运动参数检查类

以下函数在任何时候均可调用

4.1 获取各轴的逻辑位置

int get_command_pos(int cardno,int axis,long *pos)

cardno 卡号

axis 轴号

pos 逻辑位置值的指针

返回值 0：正确 1：错误

此函数可随时得到轴的逻辑位置，在电机未失步的情况下，pos **的值代表轴的当前位置。**

4.2 获取各轴的实际位置（即编码器反馈输入值）

int get_actual_pos(int cardno,int axis,long *pos)

cardno 卡号

axis 轴号(axis=1)

pos 实际位置值的指针

返回值 0：正确 1：错误

此函数可随时得到轴的实际位置，在电机有失步的情况下，pos **的值依然代表轴的实际位置。**

4.3 获取各轴的当前驱动速度

int get_speed(int cardno,int axis,long *speed)

cardno 卡号

axis 轴号

speed 当前驱动速度的指针

返回值 0：正确 1：错误

数据的单位和驱动设定数值V**一样。**

此函数可随时得到轴的驱动速度。

☞ 驱动类

5.1 定量驱动

int pmove(int cardno,int axis,long pulse)

cardno　卡号

axis　轴号

pulse　输出的脉冲数

>0：正方向移动

<0：负方向移动

范围（-268435455~+268435455）

返回值　0：正确　1：错误

注意：写入驱动命令之前一定要正确地设定速度曲线所需的参数

5.2 驱动减速停止

int dec_stop(int cardno,int axis)

cradno　卡号

axis　轴号

返回值　0：正确　1：错误

在驱动脉冲输出过程中，此命令作出减速停止，驱动速度比初始速度慢的时候也可以用本命令立即停止。

注意：直线插补时，如需要减速停止，应当只对最前的插补轴使用此指令，否则可能不能达到预定的结果。

5.3 驱动立即停止

int sudden_stop(int cardno,int axis)

cardno　卡号

axis　轴号

返回值　0：正确　1：错误

立即停止正在驱动中的脉冲输出，在加/减速驱动中也立即停止。

注意：直线插补时，如需要立即停止，应当只对最前的插补轴使用此

指令，否则可能不能达到预定的结果。

5.4 两轴直线插补

int inp_move2(int card,int axis1,int axis2,long pulse1,long pulse2)

cardno　　卡号

axis1,axis2 参与插补的轴号

pulse1,pulse2　移动的相对距离

范围（-8388608~+8388607）

返回值　0：正确　　　　1：错误

5.5 三轴直线插补

int inp_move3(int cardno, ,long pulse1,long pulse2,long pulse3)

cardno　　卡号

pulse1,pulse2,pulse3

指定轴axis1,axis2,axis3移动的相对距离

范围（-8388608~+8388607）

返回值　0：正确　　　　1：错误

☞ 开关量输入输出类

6.1 读单个输入点

int read_bit(int cardno,int number)

cardno　　卡号

number　　输入点（0-11）

返回值　0：低电平

1：高电平

-1：错误

6.2 输出单点

int write_bit(int cardno,int number,int value)

cardno　　**卡号**

number　　**输出点**（0-5）

value　　0：**低**　　　　1：**高**

返回值　　0：**正确**

1：**错误**

输出数number**对应相应的输出号。**

6.3　DA输出

int set_daout(int card,int ch,int value)

cardno　　**卡号**

ch　　**端口号**(1,3)

value　　DA**输出值**(-32768,32767)

注：value为-32768输出为-5V，value为0输出为0V，value为32767输出为5V，其余依次类推。

6.4 PWM输出

int set_pwm (int card,long freq,float value)

cardno　　**卡号**

freq　　**脉冲频率**(250-100000)

value　　**占空比**(0-1)

注：占空比为浮点数，0.5为50%的占空比，1为100%的占空比，0为0%的占空比。

☞ 延时类

7.1 延时时间

int　set_delay_time(int cardno,long time)

cardno　　**卡号**

time　　**延时时间**

返回值　　0：**正确**

1：**错误**

说明：　　**时间单位为**1/8us，**最大值为长整型的最大值。**

4 延时状态

int get_delay_status(int cardno)

cardno　　**卡号**

返回值　　0**：延时结束**

　　　　　　1**：延时进行中**

第八章 运动控制函数库使用导航

1. ADT8933 函数库概述

ADT8933 函数库实质是用户操作运动控制卡的接口,用户通过调用接口函数，即可控制运动控制卡完成相应的功能。

运动控制卡提供了 DOS 下的运动函数库和 Windows 下的动态链接库，下面分别介绍 DOS 和 Windows 下的函数库的调用方法。

2. Windows 下动态链接库的调用

Windows 下的动态链接库“ Adt8933.dll ”利用 VC 编写而成，位于光盘“ 开发包\驱动\动态链接库 ”下，适用于 Window 下常用的编程语言工具：VB、VC、C++Builder、VB.NET、VC.NET、Delphi 和组态软件 LabView 等。

2.1 VC 中的调用

(1) 新建一个项目；

(2) 将光盘“ 开发包\VC ”下的“ Adt8933.lib ”和“ Adt8933.h ”文件拷贝到新建项目的路径下；

(3) 在新建项目”工作区”的”文件视图”中，右击鼠标，选择“ Add Files to Project ”，在插入文件对话框中，文件类型选择为‘ Library Files(.lib) ”，搜索出“ Adt8933.lib ”并且选择，点击“OK ”，完成静态库的加载；

(4) 在源程序文件或头文件或全局头文件“ StdAfx.h ”的申明部分加上 #include “ Adt8933.h “ ；

经过上述四步，用户即可调用动态链接库中的函数。

说明：VC.NET 中的调用方法和 VC 相似。

2.2 VB 中的调用

(1) 新建一个项目；

(2) 将光盘“ 开发包\VB ”下的“ adt8933.bas ”文件拷贝到新建项目的路径下；

(3) 选择“ 工程\添加模块 ”菜单命令，选择对话框中的“ 现存 ”标签页，搜索出“ adt8933.bas ”模块文件，点击打开按钮；

经过上述三步，即可在程序中调用动态链接库的函数。

说明：VB.NET 中的调用方法和 VB 相似。

2.3 C++Builder 中的调用

(1) **新建一个项目；**

(2) **将光盘中"开发包**\C++Builder**"中的"**adt8933.lib**"和"**adt8933.h**"拷贝到新建项目路径下；**

(3) **选择"**Project\Add to Project**"菜单命令，在对话框中，文件类型选择为"**Library files(*.lib)**"，搜索出"**adt8933.lib**"文件，点击"打开"按钮；**

(4) **在程序文件的申明部分加上**#include "adt8933.h"**；**

经过上述三步，即可在程序中调用动态链接库。

2.4 LabView 8 中的调用

(1) **新建一个** VI**；**

(2) **将光盘中"开发包\驱动\动态链接库"中"**adt8933.dll**"拷贝到新建路径下；**

(3) **在需要调用库函数的地方，在程序框图的窗口中，在函数模板中选择"**Connectivity\Libraries & Executables**"下面的"**Call Library Function Node**"节点，添加到调用处；**

(4) **双击节点，首先在"**Call Library Function**"对话框中选择"**adt8933.dll**"动态链接库，其次选择需要的库函数，最后配置好函数的返回值和参数属性；**

经过上述四步，即可在程序中调用动态链接库。

3. DOS 下库函数的调用

DOS **下的函数库是利用** Borland C3.1 **编译而成，存放在光盘"开发包**\BC **或** C**"下，库函数分为大模式和巨模式两种，适用于标准** C **和** Borland C3.1 **或以上版本。**

Borland C **调用函数库的方法如下：**

(1) **在** Borland C **的开发环境下，选择"**Project\Open Project**"命令新建一个项目；**

(2) **将光盘中"开发包**\BC**"下面的"**ADT8933H.LIB**"或**

“ADT8933L.LIB”和“ADT8933.H”文件拷贝到新建项目路径下；

(3) 选择“Project\Add Item”命令，在对话框中选择“ADT8933H.LIB”或“ADT8933L.LIB”，单击“Add”按钮；

(4) 在用户程序文件中增加#include “adt8933.h”申明；

经过上述四步，即可在程序中调用库函数。

4. 库函数返回值及其含义

为了保证用户在使用库函数时，正确掌控库函数的执行情况，函数库中的每个库函数都会在执行结束后，返回库函数的执行结果。用户依据返回值，可以很方便地判断出函数调用是否成功。

函数库中除“int adt8933_initial(void)”和“int read_bit(int cardno, int number)”的返回值特殊外，其他函数的返回值只有“0”和“1”两种情况，其中“0”表示调用正确，“1”表示调用失败。

下面以列表的形式介绍函数返回值的含义。

函数名	返回值	含义
adt8933_initial	-1	未安装端口驱动程序
	-2	PCI 插槽故障
	0	没有安装控制卡
	>0	代表控制卡的数量
read_bit	0	低电平
	1	高电平
	-1	代表卡号或输入点超限错误
其他所有函数	0	正确
	1	错误

说明：返回值 1 错误，正常是由于调用库函数的过程中，传递的参数值 cardno(卡号)或 axis(轴号)错误引起的。卡号的值从 0、1、2 依次向上编号，所以在只用一块卡的情况下，卡号必须为 0；轴号的值只能是 1、2、3，其他的值都是错误的。

第九章 运动控制开发要点

本卡在编程时常会遇到一些问题，其实，大部分问题是由于对本控制卡的原理不理解而产生的，以下就一些常见的、易产生误解的情况作一些说明。

☞ 卡的初始化

在程序的开始首先应调用adt8933_initial()**函数，确认**adt8933**卡的安装是否正确，然后设置脉冲输出的模式，限位开关的工作模式，以上参数应根据具体的机器来设置，一般只应在程序初始化时设置一次，以后不应再设置。**

说明：库函数“ adt8933_initial ”是通往ADT8933卡的“ 门户 ”，只有在调用该函数对运动控制卡初始化成功后，再调用其他函数才有意义。

☞ 速度的设定

2.1 匀速运动

参数的设置很简单，只需要将驱动速度设置成等于起始速度，其余的参数不用设置。

相关函数：

set_startv

set_speed

2.2插补速度

adt8933**卡可以实现任意**2**轴的直线插补，以及**3**轴直线插补。**

关于插补的速度，是使用轴号为最小的轴的速度参数，作为长轴的速度，例如

inp_move2 (0,3,1,100,200)

是采用第一个轴的速度参数，即X**轴，而与参数中的顺序无关。**

inp_move2 (0,2,3,,200,500)

是采用第二个轴的速度参数，即Y轴，而与参数中的顺序无关。

说明：插补时的速度是单轴运动时速度的一半，即在同样的参数时，插补的速度只有单轴运动的一半。

第十章 运动控制开发编程示例

所有运动控制函数均为立即返回，当驱动命令发出后，运动过程由运动控制卡控制完成，此时用户的上位机软件既可以对整个运动过程进行实时监控，也可强制停止运动过程。
说明：轴在运动过程中，不允许向运动轴发新的驱动指令，否则会放弃上次的驱动，而执行后面的驱动指令。

尽管编程语言“五花八门”，种类繁多，但就其本质而言，最终可以“九九归一”。概括起来就是“三大结构和一个思想”，其中“三大结构”是指所有编程语言中都强调的顺序结构、循环结构和分支结构，一个思想主要指完成设计任务时所用到的算法以及模块划分，这是整个程序设计的重点和难点。

为了保证程序具有通用性、规范性、可扩展性以及维护方便等特点，下面所有的示例从项目设计的角度着眼，将示例划分为以下几个模块：运动控制模块(对控制卡提供的库函数进一步进行封装)，功能实现模块(配合具体工艺的代码段)，监控模块和停止处理模块。

下面我们简单介绍ADT8933卡函数库在VB和VC编程语言中的应用,如果使用其它编程语言可参照VB和VC示例程序。

☞ VB 编程示例

1.1 准备工作

(1) 新建一个项目，保存为“test.vbp ”;

(2) 按照前面讲述的方法，在项目中添加“adt8933.bas ”模块；

1.2 运动控制模块

(1) 在项目中添加一个新模块，保存为“ctrlcard.bas ”;

(2) 在运动控制模块中首先自定义运动控制卡初始化函数，对需要封装

到初始化函数中的库函数进行初始化；

(3) 继续自定义相关的运动控制函数，如：速度设定函数，单轴运动函数，差补运动函数等；

(4) ctrcard.bas 的源代码为：

```
'*********************** 运动控制模块 ********************
    '为了简单、方便、快捷地开发出通用性好、可扩展性强、
    '维护方便的应用系统，我们在控制卡函数库的基础上将
    '所有库函数进行了分类封装。下面的示例使用一块运动
    '控制卡
'********************************************************
Public Result As Integer
Const MAXAXIS = 3            '最大轴数
'******************初始化函数***********************
    '该函数中包含了控制卡初始化常用的库函数，这是调用
    '其他函数的基础，所以必须在示例程序中最先调用
    '返回值<=0 表示初始化失败，返回值>0 表示初始化成功
'****************************************************
Public Function Init_Card() As Integer
    Result = adt8933_initial              卡初始化函数
   If Result <= 0 Then
      Init_Card = Result
      Exit Function
    End If
    For   i = 1 To MAXAXIS
         set_limit_mode 0, i, 0, 0, 0     设定限位模式
         set_command_pos 0, i, 0         清逻辑计数器
          set_startv 0, i, 1000          设定起始速度
         set_speed 0, i, 1000            设定驱动速度
    Next I
    set_actual_pos 0, 1, 0           清实位计数器
```

```
    Init_Card = Result
End Function

'**********************设置速度模块***********************
    '依据参数的值，判断是匀速还是加减速
    '返回值=0 正确，返回值=1 错误
'*********************************************************
Public Function Setup_Speed(ByVal axis As Integer, ByVal StartV As Long,
    ByVal Speed As Long, ByVal Add As Long) As Integer
    If (StartV - Speed >= 0) Then
        Result = set_startv(0, axis, StartV)
        set_speed 0, axis, StartV
    Else
        Result = set_startv(0, axis, StartV)
        set_speed 0, axis, Speed
        set_acc 0, axis, Add /125
    End If
    Setup_Speed = Result
End Function

'*********************单轴驱动函数**********************
    '该函数用于驱动单个运动轴运动
    '返回值=0 正确，返回值=1 错误
'*******************************************************
Public Function Axis_Pmove(ByVal axis As Integer, ByVal value As Long) As
  Integer
    Result = pmove(0, axis, value)
    Axis_Pmove = Result
End Function
```

```
'*******************任意两轴插补函数********************
    '该函数用于驱动任意两轴进行插补运动
    '返回值=0 正确，返回值=1 错误
'*******************************************************
Public Function Interp_Move2(ByVal axis1 As Integer, ByVal axis2 As
  Integer, ByVal value1 As Long, ByVal value2 As Long) As Integer
   Result = inp_move2(0, axis1, axis2, value1, value2)
   Interp_Move2 = Result
End Function

'*******************三轴插补函数**********************
    '该函数用于驱动任意三轴进行插补运动
     '返回值=0 正确，返回值=1 错误
'*********************************************************
Public Function Interp_Move3(ByVal value1 As Long, ByVal value2 As Long,
  ByVal value3 As Long) As Integer
   Result = inp_move3(0, value1, value2, value3)
    Interp_Move3 = Result
End Function
'*****************获取运动信息****************************
    '该函数用于反馈轴当前的逻辑位置，实际位置和运行速度
    '返回值=0 正确，返回值=1 错误
'***********************************************************
Public Function Get_CurrentInf(ByVal axis As Integer, value() As Long) As
  Integer
    Result = get_command_pos(0, axis, value(0))
   If axis = 1 Then
        get_actual_pos 0, axis, value(1)
    Else
        value(1) = 0
```

```
        End If
        get_speed 0, axis, value(2)
        Get_CurrentInf = Result
    End Function

    '*******************停止轴驱动函数**************************
        '该函数提供立即停止模式和减速停止模式
        '返回值=0 正确，返回值=1 错误
    '************************************************************
    Public Function StopRun(ByVal axis As Integer, ByVal mode As Integer)
        If mode = 0 Then
            Result = sudden_stop(0, axis)
        Else
            Result = dec_stop(0, axis)
        End If
        StopRun = Result
    End Function
    '**********************清空计数器函数**********************
        '清空三轴的逻辑计数器
        '清空一路实位计数器
'**************************************************************
    Public Sub Clear_Count()
        For i = 1 To 3
            set_command_pos 0, i, 0
        Next i
        set_actual_pos 0, 1, 0
    End Sub
```

1.3 功能实现模块

1.31 界面设计

说明：

(1) **速度设定部分—用于设定各轴的起始速度、驱动速度和加速度；位置设定—设定各轴的驱动脉冲；驱动信息—实时显示各轴的逻辑位置、实际位置和运行速度。**

(2) **驱动对象—通过选择驱动对象，确定参与联动或插补的轴；**

(3) **联动—用于向所选驱动对象的所有轴发出单轴驱动指令；插补—用于向所选驱动对象的所有轴发出插补指令；停止—停止所有轴的脉冲输出；**

以上所有数据均以脉冲为单位。

1.3.2 **初始化代码位于窗体加载事件中，代码如下：**

```
Private Sub Form_Load()
    If Init_Card <= 0 Then
       MsgBox "控制卡初始化失败！"
        End
    Else
```

```
    MsgBox "运动控制卡可以使用！"
 End If
 For i = 0 To 2
     StartV(i).Text = 100
     Speed(i).Text = 200
     Add(i).Text = 100
 Next i
 For i = 0 To 2
     Pos(i).Text = 10000
 Next i
 DA1.Text = 3000
 DA2.Text = 0
 DA3.Text = -3000
 Freq1.Text = 2000
 Freq2.Text = 0.2
End Sub
```

1.3.3 **联动代码位于**CmdPmove**按钮的点击事件中，其中依据选择对象的不同发出对应的驱动指令，三个复选框(选择对象)名称分别为**X、Y、Z**，代码如下：**

```
Private Sub CmdPmove_Click()
 If X.value = 1 And Y.value = 1 And Z.value = 1 Then
     For i = 1 To 3
         Setup_Speed i, StartV(i - 1).Text, Speed(i - 1).Text, Add(i - 1).Text
         Axis_Pmove i, Pos(i - 1).Text
     Next i
 ElseIf X.value = 1 And Y.value = 1 Then
     For i = 1 To 2
         Setup_Speed i, StartV(i - 1).Text, Speed(i - 1).Text, Add(i - 1).Text
         Axis_Pmove i, Pos(i - 1).Text
```

```
        Next i
      ElseIf X.value = 1 And Z.value = 1 Then
        Setup_Speed 1, StartV(0).Text, Speed(0).Text, Add(0).Text

         Axis_Pmove 1, Pos(0).Text
         Setup_Speed 3, StartV(2).Text, Speed(2).Text, Add(2).Text
         Axis_Pmove 3, Pos(2).Text
     ElseIf Y.value = 1 And Z.value = 1 Then
         Setup_Speed 2, StartV(1).Text, Speed(1).Text, Add(1).Text
         Axis_Pmove 2, Pos(1).Text
         Setup_Speed 3, StartV(2).Text, Speed(2).Text, Add(2).Text
         Axis_Pmove 3, Pos(2).Text
     ElseIf X.value = 1 Then
         Setup_Speed 1, StartV(0).Text, Speed(0).Text, Add(0).Text
         Axis_Pmove 1, Pos(0).Text
     ElseIf Y.value = 1 Then
         Setup_Speed 2, StartV(1).Text, Speed(1).Text, Add(1).Text
         Axis_Pmove 2, Pos(1).Text
     ElseIf Z.value = 1 Then
         Setup_Speed 3, StartV(2).Text, Speed(2).Text, Add(2).Text
         Axis_Pmove 3, Pos(2).Text
     End If
  End Sub
```

1.3.4 **插补代码位于**CmdInp**按钮的点击事件中，其中依据选择对象的不同发出对应的驱动指令，三个复选框**(**选择对象**)**名称分别为**X、Y、Z**，代码如下：**

```
   Private Sub CmdInp_Click()
     If X.value = 1 And Y.value = 1 And Z.value = 1 Then
         Setup_Speed 1, StartV(0).Text, Speed(0).Text, Add(0).Text
          Interp_Move3 Pos(0).Text, Pos(1).Text, Pos(2).Text
```

```
    ElseIf X.value = 1 And Y.value = 1 Then
        Setup_Speed 1, StartV(0).Text, Speed(0).Text, Add(0).Text
        Interp_Move2 1, 2, Pos(0).Text, Pos(1).Text
    ElseIf X.value = 1 And Z.value = 1 Then
        Setup_Speed 1, StartV(0).Text, Speed(0).Text, Add(0).Text
        Interp_Move2 1, 3, Pos(0).Text, Pos(2).Text
    ElseIf Y.value = 1 And Z.value = 1 Then
        Setup_Speed 2, StartV(1).Text, Speed(1).Text, Add(1).Text
        Interp_Move2 2, 3, Pos(1).Text, Pos(2).Text
    End If
End Sub
```

1.4 监控模块

监控模块用于实时获取所有轴的驱动信息，显示运动信息，同时控制在驱动进行过程中，不允许发出新的驱动指令。监控模块利用定时器事件完成，代码如下：

```
Private Sub Timer1_Timer()
    Dim value(3) As Long
    Dim status(3) As Long
    '获取轴的驱动信息
    For i = 1 To 3
        Get_CurrentInf i, value
        For m = 0 To 2
            Inf((i - 1) * 3 + m).Caption = value(m)
        Next m
    Next i
    For i = 1 To 3
        get_status 0, i, status(i - 1)
    Next i
    '获取轴的驱动状态
```

```
    If status(0) = 0 And status(1) = 0 And status(2) = 0 Then
      CmdPmove.Enabled = True
      CmdInp.Enabled = True
    Else
       CmdPmove.Enabled = False
      CmdInp.Enabled = False
    End If
End Sub
```

1.5 停止模块

停止模块主要用于控制驱动过程中的突发事件，需要立即终止所有轴的驱动。停止模块的代码位于 CmdStop**按钮的点击事件中，代码如下：**

```
Private Sub CmdStop_Click()
   For i = 1 To 3
       StopRun i, 0
    Next i
End Sub
```

☞ VC 编程示例

2.1 准备工作

(1) **新建一个项目**,**保存为**“VCExample.dsw”**;**

(2) **根据前面讲述的方法，将静态库**“adt8933.lib”**加载到项目中；**

2.2 运动控制模块

(1) **在项目中添加一个新类，头文件保存为**“CtrlCard.h”,**源文件保存为**“CtrlCard.cpp”**;**

(2) **在运动控制模块中首先自定义运动控制卡初始化函数，对需要封装到初始化函数中的库函数进行初始化；**

(3) **继续自定义相关的运动控制函数，如：速度设定函数，单轴运动函数，差补运动函数等；**

(4) **头文件** “CtrlCard.h”**代码如下：**

```
# ifndef __ADT8933__CARD__
# define __ADT8933__CARD__
/*********************** 运动控制模块 ********************
    为了简单、方便、快捷地开发出通用性好、可扩展性强、
    维护方便的应用系统，我们在控制卡函数库的基础上将
    所有库函数进行了分类封装。下面的示例使用一块运动
    控制卡
********************************************************/
#define   MAXAXIS   3      //最大轴数
class CCtrlCard
{
public:
    void Set_Pwm(long freq,float value);
    void Set_DA(int ch,int value);
    int Get_Status(int axis, int &value, int mode);
    int StopRun(int axis,int mode);
    int Get_CurrentInf(int axis, long &LogPos, long &ActPos, long &Speed);
    int Interp_Move3(long value1,long value2,long value3);
    int Interp_Move2(int axis1,int axis2,long value1,long value2);
    int Axis_Pmove(int axis, long value);
    int Setup_Speed(int axis,long startv,long speed,long add);
    int Init_Board();
    void Clear_Count();
    CCtrlCard();
    int Result;
};
# endif
```

(5) **源文件 “** CtrlCard.cpp **” 代码如下：**

```
# include "stdafx.h"
# include "adt8933.h"
# include "CtrlCard.h"
# include "VCExample.h"

CCtrlCard::CCtrlCard()
{
}
/*******************初始化函数************************
     该函数中包含了控制卡初始化常用的库函数，这是调用
     其他函数的基础，所以必须在示例程序中最先调用
     返回值<=0表示初始化失败，返回值>0表示初始化成功
****************************************************/
int CCtrlCard::Init_Board()
{
        Result = adt8933_initial() ;            //卡初始化函数
        if (Result <= 0) return Result;
      for (int i = 1; i<=MAXAXIS; i++){
          set_limit_mode (0, i, 0, 0, 0);    //设定限位模式
           set_command_pos (0, i, 0);           //清逻辑计数器
           set_startv (0, i, 1000);             //设定起始速度
           set_speed (0, i, 1000);              //设定驱动速度
       }
       set_actual_pos (0, 1, 0);            //清实位计数器
      return 1;
}

/*********************设置速度模块***********************
     依据参数的值，判断是匀速还是加减速
     返回值=0正确，返回值=1错误
```

```
*********************************************************/
int CCtrlCard::Setup_Speed(int axis, long startv, long speed, long add)
{
        if (startv - speed >= 0) {
            Result = set_startv(0, axis, startv);
            set_speed (0, axis, startv);
        }
       else{
            Result = set_startv(0, axis, startv);
            set_speed (0, axis, speed);
             set_acc (0, axis, add/125);
       }
        return Result;
}

/*********************单轴驱动函数**********************
       该函数用于驱动单个运动轴运动
       返回值=0正确，返回值=1错误
*******************************************************/
int CCtrlCard::Axis_Pmove(int axis, long value)
{
        Result = pmove(0, axis, value);
        return Result;
}

/******************任意两轴插补函数*******************
       该函数用于驱动任意两轴进行插补运动
       返回值=0正确，返回值=1错误
*******************************************************/
int CCtrlCard::Interp_Move2(int axis1, int axis2, long value1, long value2)
```

```
{
        Result = inp_move2(0, axis1, axis2, value1, value2);
        return Result;
}

/*******************任意三轴插补函数***********************
        该函数用于驱动任意三轴进行插补运动
        返回值=0正确，返回值=1错误
*********************************************************/
int CCtrlCard::Interp_Move3( long value1, long value2, long value3)
{
        Result = inp_move3(0,   value1, value2, value3);
        return Result;
}

/*****************获取运动信息*****************************
        该函数用于反馈轴当前的逻辑位置，实际位置和运行速度
        返回值=0正确，返回值=1错误
**********************************************************/
int CCtrlCard::Get_CurrentInf(int axis, long &LogPos, long &ActPos, long
&Speed)
{
        get_command_pos(0, axis, &LogPos);
        if (axis==1)
                get_actual_pos (0, axis, &ActPos);
        else
                ActPos=0;
      get_speed (0, axis, &Speed);
        return 0;
}
```

```
/*****************停止轴驱动******************************
    该函数用于立即或减速停止轴的驱动
    返回值=0正确，返回值=1错误
*********************************************************/
int CCtrlCard::StopRun(int axis, int mode)
{
        if (mode = 0)
            Result = sudden_stop(0, axis);
      else
             Result = dec_stop(0, axis);
     return Result;
}

/*****************获取轴的驱动状态**************************
    该函数用于获取单轴的驱动状态或插补驱动状态
    返回值=0正确，返回值=1错误
**********************************************************/
int CCtrlCard::Get_Status(int axis, int &value, int mode)
{
        if (mode==0)
               Result=get_status(0,axis,&value);
        else
               Result=get_inp_status(0,&value);
        return Result;
}

/*******************清空计数器函数***************************
    清空三轴的逻辑计数器
    清空一路实位计数器
```

```
*************************************************************/
void CCtrlCard::Clear_Count()
{
     for (int i = 1; i<4; i++)
           set_command_pos (0, i, 0);
     set_actual_pos (0, 1, 0);
}

/*******************DA输出函数*******************************
      ch(1-3)
      value(-32768--32767)
*************************************************************/
void CCtrlCard::Set_DA(int ch, int value)
{
       set_daout(0,ch,value);
}
/*******************PWM 输出函数*******************************
      freq(250-100000)
      value(0--1)
**************************************************************/
void CCtrlCard::Set_Pwm(long freq, float value)
{
       set_pwm(0,freq,value);
}
```

2.3 功能实现模块

2.3.1 界面设计

说明：

(1)**速度设定部分—用于设定各轴的起始速度、驱动速度和加速度；位置设定—设定各轴的驱动脉冲；驱动信息—实时显示各轴的逻辑位置、实际位置和运行速度。**

(2)**驱动对象—通过选择驱动对象，确定参与联动或插补的轴；**

(3)**联动—用于向所选驱动对象的所有轴发出单轴驱动指令；插补—用于向所选驱动对象的所有轴发出插补指令；停止—停止所有轴的脉冲输出；**

以上所有数据均以脉冲为单位。

int g_CardVer=0 **定**0**义全局变量，**g_CardVer**为硬件版本。**

2.3.2 **运动控制卡初始化代码位于窗体初始化中，用户新增代码如下：**

```
if (g_CtrlCard.Init_Board() <= 0){
    MessageBox( "控制卡初始化失败!");
      return false;
}
else
```

```
MessageBox ("运动控制卡可以使用!");
UINT
    nID1[]={IDC_EDIT_STARTX,IDC_EDIT_STARTY,IDC_EDIT_STA
    RTZ};
UINT
    nID2[]={IDC_EDIT_XSPEED,IDC_EDIT_YSPEED,IDC_EDIT_ZSPE
    ED};
UINT nID3[]={IDC_EDIT_ADDX,IDC_EDIT_ADDY,IDC_EDIT_ADDZ,};
UINT nID4[]={IDC_EDIT_POSX,IDC_EDIT_POSY,IDC_EDIT_POSZ};
CEdit *pEdit;
for (int i = 0 ; i<3; i++){
    pEdit=(CEdit*)GetDlgItem(nID1[i]);
    pEdit->SetWindowText("100");
    pEdit=(CEdit*)GetDlgItem(nID2[i]);
    pEdit->SetWindowText("200");
    pEdit=(CEdit*)GetDlgItem(nID3[i]);
    pEdit->SetWindowText("100");
    pEdit=(CEdit*)GetDlgItem(nID4[i]);
    pEdit->SetWindowText("10000");
}
UpdateData(true);
m_DA1=-3000;
m_DA2=0;
m_DA3=3000;
m_freq1=2000;
m_freq2=0.2;
UpdateData(false);
SetTimer(1001,100,NULL);
```

2.3.3 **联动代码位于联动按钮点击消息中，其中依据选择对象的不同发出**

对应的驱动指令，代码如下：

```
void CVCExampleDlg::OnButtonPmove()
{
    UpdateData(TRUE);
    long startv[]={m_nStartX,m_nStartY,m_nStartZ};
    long Speed[]={m_nXSpeed,m_nYSpeed,m_nZSpeed};
    long Add[]={m_nAddX,m_nAddY,m_nAddZ};
    long Pos[]={m_nPosX,m_nPosY,m_nPosZ};
    if(m_bX && m_bY && m_bZ){
      for (int i = 1; i<4; i++){
            g_CtrlCard.Setup_Speed(i,  startv[i-1], Speed[i-1], Add[i-1]);
            g_CtrlCard.Axis_Pmove(i, Pos[i-1]);
       }
    }
    else if( m_bX && m_bY){
      for (int i = 1 ; i<3; i++){
            g_CtrlCard.Setup_Speed(i,startv[i-1],Speed[i-1],Add[i-1]);
            g_CtrlCard.Axis_Pmove(i,Pos[i-1]);
       }
    }
   else if( m_bX && m_bZ){
        g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);
        g_CtrlCard.Axis_Pmove(1,Pos[0]);
        g_CtrlCard.Setup_Speed(3,startv[2],Speed[2],Add[2]);
        g_CtrlCard.Axis_Pmove(3,Pos[2]);
    }
   else if( m_bY && m_bZ){
        g_CtrlCard.Setup_Speed(2,startv[1],Speed[1],Add[1]);
        g_CtrlCard.Axis_Pmove(2,Pos[1]);
        g_CtrlCard.Setup_Speed(3,startv[2],Speed[2],Add[2]);
```

```
                g_CtrlCard.Axis_Pmove(3,Pos[2]);
            }
           else  if(  m_bX){
                g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);
                g_CtrlCard.Axis_Pmove(1,Pos[0]);
            }
           else  if(  m_bY){
                g_CtrlCard.Setup_Speed(2,startv[1],Speed[1],Add[1]);
                g_CtrlCard.Axis_Pmove(2,Pos[1]);
            }
           else  if(  m_bZ){
                g_CtrlCard.Setup_Speed(3,startv[2],Speed[2],Add[2]);
                g_CtrlCard.Axis_Pmove(3,Pos[2]);
            }
      }
```

2.3.4 **插补代码位于插补按钮点击消息中，其中依据选择对象的不同发出对应的驱动指令，代码如下：**

```
void CVCExampleDlg::OnButtonInpmove()
{
      UpdateData(TRUE);
      long startv[]={m_nStartX,m_nStartY,m_nStartZ};
      long  Speed[]={m_nXSpeed,m_nYSpeed,m_nZSpeed};
      long Add[]={m_nAddX,m_nAddY,m_nAddZ};
      long Pos[]={m_nPosX,m_nPosY,m_nPosZ};
      if(m_bX  &&  m_bY  &&  m_bZ){
         g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);
          g_CtrlCard.Interp_Move3(Pos[0],Pos[1],Pos[2]);
      }
    else  if(m_bX  &&  m_bY){
```

```
          g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);
          g_CtrlCard.Interp_Move2(1,2,Pos[0],Pos[1]);
      }
      else  if(m_bX  &&  m_bZ){
         g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);
          g_CtrlCard.Interp_Move2(1,3,Pos[0],Pos[2]);
      }
    else  if(m_bY  &&  m_bZ){
         g_CtrlCard.Setup_Speed(2,startv[1],Speed[1],Add[1]);
          g_CtrlCard.Interp_Move2(2,3,Pos[1],Pos[2]);
      }
}
```

2.4 监控模块

监控模块用于实时获取所有轴的驱动信息，显示运动信息，同时控制在驱动进行过程中，不允许发出新的驱动指令。监控模块利用定时器消息完成，代码如下：

```
void CVCExampleDlg::OnTimer(UINT nIDEvent)
{
        long  log,act,spd;
         UINT
              nID1[]={IDC_STATIC_LOGX,IDC_STATIC_LOGY,IDC_STATIC_LOGZ};
          UINT
               nID2[]={IDC_STATIC_ACTX,IDC_STATIC_ACTY,IDC_STATIC_ACTZ};
          UINT
            nID3[]={IDC_STATIC_SPEEDX,IDC_STATIC_SPEEDY,IDC_STATIC_SPEEDZ};
        CStatic *lbl;
```

```
CString str;
int status[3];
for (int i=1; i<4; i++){
        g_CtrlCard.Get_CurrentInf(i,log,act,spd);
        str.Format("%ld",log);
        lbl=(CStatic*)GetDlgItem(nID1[i-1]);
        str.Format("%ld",log);
        lbl->SetWindowText(str);
        lbl=(CStatic*)GetDlgItem(nID2[i-1]);
        str.Format("%ld",act);
        lbl->SetWindowText(str);
        lbl=(CStatic*)GetDlgItem(nID3[i-1]);
        str.Format("%ld",spd);
        lbl->SetWindowText(str);
        g_CtrlCard.Get_Status(i,status[i-1],0);
}
CButton *btn;
if (status[0]==0 && status[1]==0 && status[2]==0)
{
        btn=(CButton*)GetDlgItem(IDC_BUTTON_PMOVE);
        btn->EnableWindow(TRUE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_INPMOVE);
        btn->EnableWindow(TRUE);
}
else
{
        btn=(CButton*)GetDlgItem(IDC_BUTTON_PMOVE);
        btn->EnableWindow(FALSE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_INPMOVE);
        btn->EnableWindow(FALSE);
```

```
        }
        CDialog::OnTimer(nIDEvent);
    }
```

2.5 停止模块

停止模块主要用于控制驱动过程中的突发事件，需要立即终止所有轴的驱动。停止模块的代码位于停止按钮的点击消息中，代码如下：

```
void CVCExampleDlg::OnButtonStoprun()
{
        for (int i = 1; i<4; i++){
                g_CtrlCard.StopRun(i,0);
        }
}
```

2.6 计数器清空模块

```
void CVCExampleDlg::OnButtonClear()
{
        g_CtrlCard.Clear_Count();
}
```

2.7 DA输出

```
void CVCExampleDlg::OnButtonDa()
{
        UpdateData(true);
        g_CtrlCard.Set_DA(1,m_DA1);
        g_CtrlCard.Set_DA(1,m_DA2);
        g_CtrlCard.Set_DA(1,m_DA3)
}
```

2.8 PWM输出

```
void CVCExampleDlg::OnButtonPwm()
{
        UpdateData(true);
        g_CtrlCard.Set_Pwm(m_freq1,m_freq2);
}
```

第十一章 常见故障及解决方案

☞ 运动控制卡检测失败

在使用控制卡的过程中，如果遇到检测不到控制卡的现象，可以参照下面的方法逐步进行排查。

(1) **务必按照控制卡安装说明,分步安装好控制卡的驱动程序,在**Win2000**和**WinXP**下必须安装端口驱动,确保在系统目录（**system32**或**System**）下有控制卡动态库文件；**

(2) **检查运动控制卡和插槽接触是否良好。可以通过重插或更换插槽的方式测试，另可用橡皮擦对控制卡的金手指的污诟进行清除，再装上测试；**

(3) **在系统设备管理器中，检查运动控制卡和其它硬件是否有冲突。使用**PCI**卡时，可以先取下其它板卡，如：声卡，网卡等；**PC104**卡可以调整拨码开关重新设定基地址，程序中卡初始化时使用的基地址必须和实际基地址相同；**

(4) **检查操作系统是否存在问题，可以通过重新安装其他版本的操作系统进行测试；**

(5) **按照上面的步骤检查后，如果依然找不到运动控制卡，可以通过更换运动控制卡，进一步进行检测，以便诊断运动控制卡是否已经损坏；**

☞ 电机运行异常

在运动控制卡正常的前提下，电机出现异常现象时，可以参照下面的情况排除故障。

(1) **运动控制卡发出脉冲时，电机不运动**

- **请检查控制卡和端子板的连接线是否接好；**
- **电机驱动器的脉冲和方向信号线是否已经正确地连接到端子板；**

- 伺服驱动器的外部电源是否已经连接好；
- 伺服、步进电机驱动器是否存在报警状态,如有报警则按报警对应代码检查原因所在；
- 伺服SON是否连接好，伺服电机是否有激磁状态等；
- 如果是伺服电机请检查驱动器的控制方式，本公司的控制卡支持“ 位置控制方式 ”;
- 电机、驱动器坏

(2) 步进电机运转时发出异常尖叫声，电机出现明显失步现象。

- 控制器的速度过快，请计算电机的速度，步进电机每秒10~15转属于正常范围；
- 机械部件卡死或机器的阻力太大；
- 电机的选型不够，请更换大力矩型号电机；
- 请检查驱动器的电流和电压，电流设为电机的额定电流的1.2倍,供电电压在驱动器的额定范围；
- 检查控制器起始速度，一般起始速度为0.5~1左右，加减速时间0.1秒以上；

(3) 伺服、步进电机在加工过程中出现明显振动或噪音现象

- 伺服驱动器的位置环增益和速度环增益太大，在定位精度允许的情况下降低服驱动器的位置环增益和速度环增益；
- 机器刚性太差，调整机器的结构；
- 步进电机选型不够，请更换大力矩型号电机；
- 步进电机的速度处于电机的共振区域，请避开此共振区或增大细分的办法来解决；

(4) 电机定位不准

- 请检查机械丝杠螺距和电机每转脉冲数与实际应用系统所设定的参数是否相符，即脉冲当量；
- 如果伺服电机，则增大位置环增益和速度环增益；

- 请检查机器的丝杠间隙，用千分表测试丝杠的反向间隙，如有间隙请调整丝杠；
- 如果是不定时、不定位置的定位不准，则要检查外部干扰信号；
- 电机选型不够在运动中出现抖动或失步现象；

(5) 电机没有方向

- 检查DR+　DR-接线有没有错误，是否接牢；
- 请确定控制卡采用的脉冲模式是否与实际驱动器模式相符，本控制卡支持“脉冲+方向”和“脉冲+脉冲模式”
- 步进电机要检查电机线有没断线、接触不良等现象；

☞ 开关量输入异常

在系统调试、运行过程中，某些输入信号检测异常，可以使用下面介绍的方法进行检查。

(1) 没有信号输入

- 据前面讲述的普通开关和接近开关的接线图，检查线路是否正确，确保输入信号的“光藕公共端”已经和内部或外部电源(+12V或24V)的正端相连；
- 本公司的I\O点的输入开关使用NPN型，如果没有请检查开关开型号和接线方式；
- 检查光藕是否已经损坏。在线路正常的情况下，输入点在断开和闭合的情况下，输入状态不发生改变，可以利用万用表检测光藕是否已经被击穿，通过更换光藕可以解决光藕被击穿的问题；
- 检查开关电源的12V或24V是否正常；
- 开关损坏；

(2) **信号时有时无**

- **检查是否存在干扰，可以在I\O测试画面检测信号的状态；如果是干扰情况则增加独石电容型号为104或采用屏蔽线等；**
- **机械在正常运行过程中，出现明显的颤抖或异常停止现象，请检查限位开关信号是否存在干扰或限位开关性能是否可靠；**
- **外部接线是否接触良好；**

(3) **归零不准**

- **速度太快，降低归零速度；**
- **外部信号存在干扰，请检查干扰源；**
- **归零方向错误；**
- **归零开关安装位置不当或开关松动；**

(4) **限位无效**

- **在I\O测试下检测限位开关是否有效；**
- **手动、自动加工时速度太快；**
- **外部信号存在干扰，请检查干扰源；**
- **手动方向错误；**
- **限位开关安装位置不当或开关松动；**

☞ 开关量输出异常

开关量输出异常，可以依据下面介绍的方法进行排查。

(1) **输出异常**

- **依据前面讲述的输出点的接线图，检查线路是否正确，确保输出公共端(地线)和所用电源的地线相连；**
- **检查输出器件是否已经损坏；**

- ➢ **检查光藕是否已经损坏,利用万用表检测光藕是否已经被击穿，通过更换光藕可以解决光藕被击穿的问题；**
- ➢ **安全要领。输出使用感性负载时一定要并联续流二级管，型号**IN4007**或**IN4001**；**

(2) **输出不良判断方法**

断开输出点上对外的接线，在输出点上接一10K**左右的上拉电阻到电源端，此时输出的地线需接到电源的**GND**，并用万用表的红表笔点有**12V**的正极，黑表笔点在信号输出端同时用手点动测试画面的按钮看是否有电压输出，如果有则检查外围线路，否则检查板卡的公共端是否接好、内部光藕不良等；**

☞ 编码器异常

在使用编码器出现异常时，可以参照下面介绍的方法进行排查。

(1) **检查编码器接线。确保编码器的接线符合前面介绍的差动或集电极开路方式；**

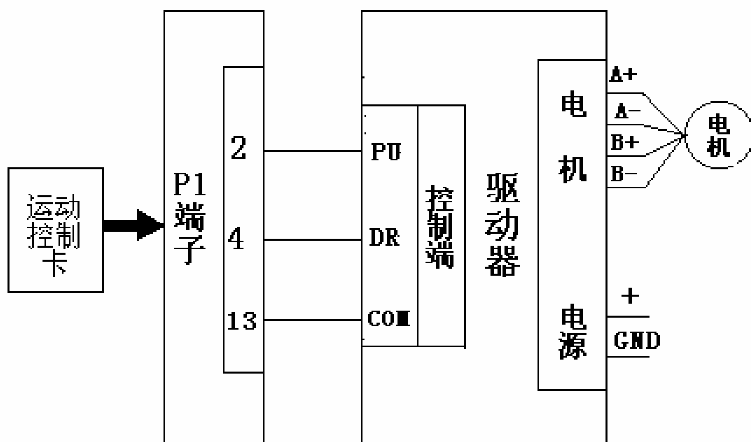
(2) **检测编码器电压。运动控制卡正常接受的是**+5V**的信号，如果选用的是**+12V**或**+24V**编码器，务必在编码器**A、B**相和端子板**A、B**相之间串联**1K(+12V)**电阻；**

(3) **编码器计数不准。编码器的外围接线一定要采用屏蔽双绞线，编码器线不能跟强电等一些干扰较强的电线捆绑在一起，必须分开在**30～50MM**以上；**

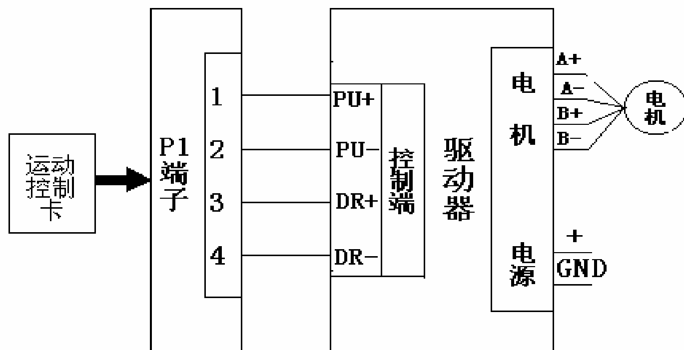
附录A 电机驱动器典型接线图

下面的接线都以X轴为例。

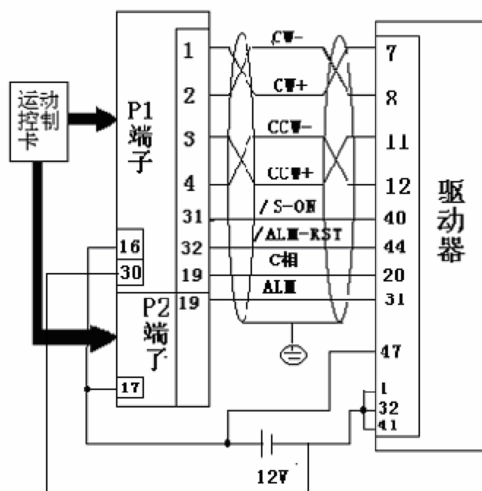
☞ **步进电机驱动器共阳极接线图**



☞ **步进电机驱动器差动接线图**



☞ 安川伺服驱动器接线图



☞ 松下A4伺服驱动器接线图

